SMS Verify - Tutorial: Send one-time passcode with Telesign Java SDK

This tutorial teaches you how to use the Telesign Java SDK to send an SMS with a one-time passcode (OTP). Go to GitHub to see the complete sample code.

Before you begin

Make sure you have the following before you start:

- **Authentication credentials:** Your Customer ID and API Key. If you need help finding these items, go to the support article How do I find my Customer ID and API Key.
- **Testing device:** A mobile phone on which you can receive SMS.
- **Gradle:** This package manager isn't required to use the SDK, but it is required for this tutorial.

NOTE:
This tutorial uses the following:

- MacOS
- Java OpenJDK v19.0.2
- Gradle 8.0.2

Please modify accordingly if your developer environment differs from these details.

Set up your project

CAUTION
You should use the Full-service SDK for SMS Verify even if you have a Self-service account!

1. Follow the Telesign Full-service Java SDK install instructions on GitHub here, incorporating the following details:
    - Use `sms_verify` as the project directory name.
    - Use `JUnit 4` as the test framework.
    - Use `sendOTP` as the project name.

- Use `sendOTP` as the source package name.

You should end up in the top-level directory ("sms_verify") for your project in the Terminal. This directory should contain an initialized Gradle project with the Telesign Self-service SDK included in its dependencies and .java source files from the Telesign Full-service SDK copied into it.

2. From the top level of your project, delete the folder "app/src/test/". No tests are included in this tutorial.

   Terminal

   ```
   rm -r app/src/test/
   ```

3. Open the file "build.gradle.kts". At the bottom of this file, add the following declaration.

   Java

   ```
   tasks.named<JavaExec>("run") {
       standardInput = System.`in`
   }
   ```

   This is needed to enable the Scanner utility to work properly with Gradle. We'll be using Scanner to collect input on the command line.

Create code to send the SMS

4. Open the file "app/src/main/java/sendOTP/App.java".

5. Add the imports below between the package declaration and the App class declaration:

   Java

   ```
   import com.telesign.RestClient;
   import com.telesign.Util;
   import com.telesign.enterprise.VerifyClient;
   import java.util.HashMap;
   import java.util.Scanner;
   ```

   These imports reference selected functionality from the Telesign Self-service and Full-service SDKs, as well as some other Java utilities.

6. Replace the the default App class with the following basic structure.

   Java

   ```
   public class App {
       public static void main(String[] args) {

       }
   }
   ```

   In the next step, you will begin adding statements to the main function in the App class.

7. Define variables in the main function to store your authentication credentials. For testing purposes, you can just overwrite the default values below or use environment variables.

   Java

   ```
   String customerId = System.getenv().getOrDefault("CUSTOMER_ID",
   "FFFFFFFF-EEEE-DDDD-1234-AB1234567890");
   String apiKey = System.getenv().getOrDefault("API_KEY",
   "ABC12345yusumoN6BYsBVkh+yRJ5czgsnCehZaOYldPJdmFh6NeX8kunZ2zU1YWaUw/
   0wV6xfw==");
   ```

8. Define a variable to hold the recipient's phone number. For this tutorial, hardcode your testing device's phone number or pull it from an environment variable.

   Java

   ```
   String phoneNumber = System.getenv().getOrDefault("PHONE_NUMBER",
   "11234567890");
   ```

   NOTE:
   In your production integration, pull the phone number from your recipient database instead of hardcoding it.

9. Randomly generate your OTP. We will use a Telesign SDK utility for this. The parameter value 5 specifies the number of digits generated:

   Java

   ```
   String verifyCode = Util.randomWithNDigits(5);
   ```

CAUTION
The method used above to generate a code is actually *pseudo-random*. In your production implementation, you might want to use a more robust method for randomizing.

10. Create a new hash map, and use it to store the params you are going to send to the Telesign Verify API. The only param included in this tutorial is the verification code.

    Java

    ```
    HashMap<String, String> params = new HashMap<>();
    ```

11. Create a try-catch structure.

    Java

    ```
    try {

    } catch (Exception e) {

    }
    ```

12. In the try block, add code to instantiate a Telesign verification client object with your authentication credentials.

    Java

    ```
    VerifyClient verifyClient = new VerifyClient(customerId, apiKey);
    ```

    NOTE:
    When you use a Telesign SDK to make your request, authentication is handled behind-the-scenes for you. All you need to provide is your Customer ID and API Key. The SDKs apply Digest authentication whenever they make a request to a Telesign service where it is supported. When Digest authentication is not supported, the SDKs apply Basic authentication.

13. Next in the `try` block, make the request and capture the response. Behind the scenes, this sends an HTTP request to the Telesign Verify API. Telesign then sends an SMS with an OTP to the end-user:

    Java

    ```
    RestClient.TelesignResponse telesignResponse =
    verifyClient.sms(phoneNumber, params);
    ```

14. Next in the try block, display the response in the console for debugging purposes. In your production code, you would likely remove this.

    Java

    ```
    System.out.println("\n" + "Response HTTP status:" +
    telesignResponse.statusCode);
    System.out.println("Response body:" + telesignResponse.body + "\n");
    ```

15. Next in the try block, collect the asserted OTP entered by the end-user in your application. You can simulate this by prompting for input from the command line:

    Java

    ```
    System.out.println("Please enter the verification code you were
    sent:");
    Scanner s = new Scanner(System.in);
    String code = s.next();
    ```

    NOTE:
    In your production implementation, collect input from your website or other application where the end-user is trying to log in.

16. Next in the try block, determine if the user-entered code matches your OTP, and resolve the login attempt accordingly. You can simulate this by reporting whether the codes match.

    Java

    ```
    if (verifyCode.equalsIgnoreCase(code)) {
      System.out.println("Your code is correct.");
    } else {
      System.out.println("Your code is incorrect.");
    }
    ```

    NOTE:
    In your production implementation, add code here to log in the user when the user-entered code matches the OTP.

17. In the catch block, add code to print any error that might occur.

    Java

    ```
    System.out.println((char)27 + "[31m" + "\nAn exception
    occurred.\nERROR: " + e.getMessage());
    ```

Test your integration

18. Switch from your editor to the terminal and build your project. Make sure you are in the top-level directory of your project ("sms_verify").

    Terminal

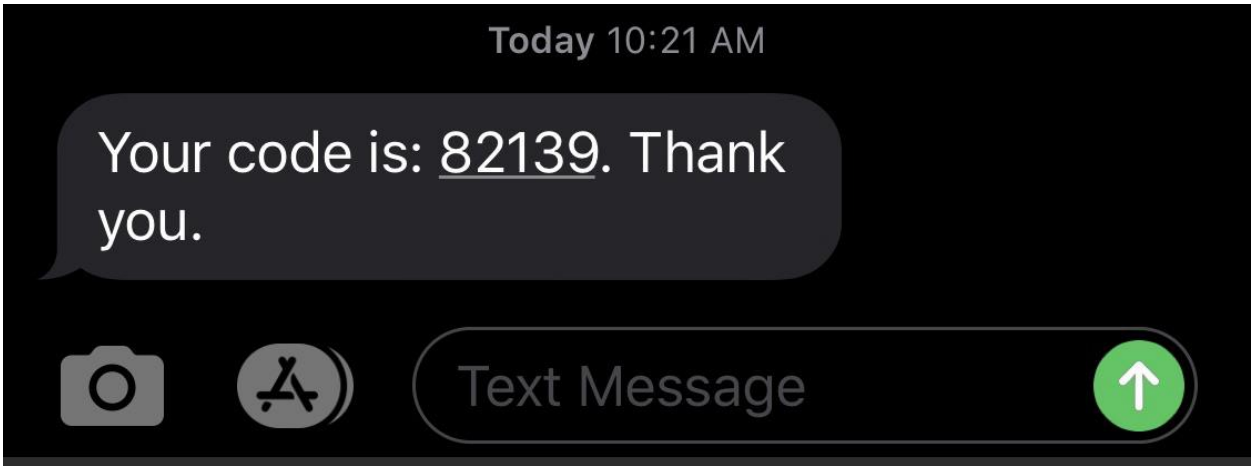
    ```
    ./gradlew build
    ```

19. Run your project.

    Terminal

    ```
    ./gradlew run
    ```

    You should receive an SMS on your phone that looks like this:



20. Enter the OTP you received on your phone at the command prompt on the terminal to test that verification is successful:

    Terminal

    ```
    Please enter the verification code you were sent: 82139
    Your code is correct.
    ```

21. Now let's test an unsuccessful verification. Run again.

    Terminal

    ```
    ./gradlew run
    ```

You should receive a new OTP on your phone.

22. Enter something else that isn't correct at the command prompt on the terminal and you should get a message that verification failed:

    Terminal

    ```
    Please enter the verification code you were sent: 55555
    Your code is incorrect.
    ```

Sample code

The complete sample code for this tutorial can be found on GitHub.